

14

Hyperledger

Hyperledger is not a blockchain but a project that was initiated by the Linux Foundation in December 2015 to advance blockchain technology. This project is a collaborative effort by its members to build an open source distributed ledger framework that can be used to develop and implement cross-industry blockchain applications and systems. The principal focus is to create and run platforms that support global business transactions. The project also focuses on improving the reliability and performance of blockchain systems.

This chapter will, for the most part, discuss Hyperledger projects, and their various features and components. As Hyperledger is so vast it's not possible to cover all projects in detail in this short chapter. As such, we will introduce a variety of the projects available, and then provide detailed discussions on Hyperledger Fabric. Projects under the Hyperledger umbrella undergo multiple stages of development, going from proposal, to incubation, and eventually graduating to an active state. Projects can also be deprecated or put in an end-of-life state where they are no longer actively developed. For a project to be able to move into the incubation stage, it must have a fully working code base along with an active community of developers. The Hyperledger project currently has more than 300 member organizations and is very active, with many contributors and meet-ups and talks organized around the globe.

We will first cover some of the various projects under Hyperledger. Then we will move on to examine the design and architecture of Hyperledger Fabric in more detail, covering the following topics on the way:

- Projects under Hyperledger
- Hyperledger reference architecture
- Hyperledger Fabric
- Fabric 2.0

Projects under Hyperledger fall into different categories, and we'll start with the introduction of these categories in the next section.

Projects under Hyperledger

There are four categories of projects under Hyperledger. Under each category, there are multiple projects. The categories are:

- Distributed ledgers
- Libraries
- Tools
- Domain-specific

We will look at five distributed ledger projects under the Hyperledger umbrella: **Fabric**, **Sawtooth**, **Iroha**, **Indy**, and **Besu**. Under libraries, we will explore the **Aries**, **Transact**, **Ursa**, and **AnonCreds** projects. Under the tools category we will look at projects such as **Cello**, **Caliper**, along with a domain-specific project called Hyperledger **Grid**. There are various other projects underway, (they can be reviewed at <https://www.hyperledger.org>) but for brevity, we are including only the more relevant and interesting here.

Also note that this landscape keeps changing due to developments and improvements, which you can keep an eye on here: <https://landscape.hyperledger.org/>

A brief introduction to all these projects follows.

Distributed ledgers

Distributed ledgers, as we covered in *Chapter 1, Blockchain 101*, are a broad category of distributed databases that are decentralized, shared among multiple participants, and updateable only via consensus.

Often, the terms *blockchain* and *distributed ledger* are used interchangeably. However, one key difference between distributed ledgers and blockchains is that blockchains are expected to have an architecture in which transactions are handled in batches of **blocks**, whereas distributed ledgers have no such requirements. All other attributes of distributed ledgers and blockchains are broadly the same. In Hyperledger, *distributed ledger* is a generic term used to denote both blockchains and distributed ledgers.

Fabric

Hyperledger Fabric is a blockchain project that was proposed by **IBM** and **Digital Asset Holdings (DAH)**. It is an enterprise-grade permissioned distributed ledger framework for the development of blockchain solutions and applications. Fabric is based on a modular and pluggable architecture. This means that various components, such as the consensus engine and membership services, can be plugged into the system as required. Currently, its status is **active** and it is the first project to graduate from **incubation** to **active** state. We'll cover Fabric in more detail later in this chapter.



The source code is available at <https://github.com/hyperledger/fabric>

Sawtooth

Hyperledger Sawtooth is a blockchain project proposed by Intel in April 2016. It was donated to the Linux Foundation in 2016. Sawtooth introduced some novel innovations focusing on the decoupling of ledgers from transactions, flexible usage across multiple business areas using transaction families, and pluggable consensus. It is an enterprise-grade blockchain with a focus on privacy, security, and scalability.

Ledger decoupling can be explained more precisely by saying that the transactions are decoupled from the consensus layer, by making use of a new concept called **transaction families**. Instead of transactions being individually coupled with the ledger, transaction families are used, which allows more flexibility, rich semantics, and the open design of business logic. Transactions follow the patterns and structures defined in the transaction families.

One of the innovative features that Intel has introduced with Hyperledger Sawtooth is a novel consensus algorithm called **Proof of Elapsed Time (PoET)**. It makes use of the **Trusted Execution Environment (TEE)** provided by **Intel Software Guard Extensions (Intel SGX)** to provide a safe and random leader election process. It supports both permissioned and permissionless setups.



This project is available at <https://github.com/hyperledger/sawtooth-core>

Iroha

Iroha was contributed by Soramitsu, Hitachi, NTT Data, and Colu in September 2016. Iroha aims to build a library of reusable components that users can choose to run on their own Hyperledger-based distributed ledgers.

Iroha's primary goal is to complement other Hyperledger projects by providing reusable components written in C++ with an emphasis on mobile development. This project has also proposed a novel consensus algorithm called **Sumeragi**, which is a chain-based Byzantine fault-tolerant consensus algorithm.



Iroha is available at <https://github.com/hyperledger/iroha>

Various libraries have been proposed and are being worked on by Iroha, including, but not limited to, a digital signature library (ed25519), a SHA-3 hashing library, a transaction serialization library, a P2P library, an API server library, an iOS library, an Android library, and a JavaScript library.

Indy

This project has graduated status under Hyperledger. Indy is a distributed ledger developed for building decentralized identities. It provides tools, utility libraries, and modules, which can be used to build blockchain-based digital identities. These identities can be used across multiple blockchains, domains, and applications. Indy has its own distributed ledger and uses **Redundant Byzantine Fault Tolerance (RBFT)** for consensus.



The source code is available at <https://github.com/hyperledger/indy-node>

Besu

Besu is a Java-based Ethereum client. It is the first project under Hyperledger that can operate on a public Ethereum network.



The source code of Besu is available at <https://github.com/hyperledger/besu>

With this, we have completed an introduction to distributed ledger projects under Hyperledger. Let's now look at the next category, libraries.

Libraries

A number of libraries are currently available under the Hyperledger project. This category includes projects that aim to support the blockchain ecosystem by introducing platforms for interoperability, identity, cryptography, and developer tools. We will now briefly describe each one of these.

Aries

Aries is not a blockchain; in fact, it is an infrastructure for blockchain-rooted, **peer-to-peer (P2P)** interactions. The aim of this project is to provide code for P2P interactions, secrets management, verifiable information exchange (such as verifiable credentials), interoperability, and secure messaging for decentralized systems. The eventual goal of this project is to provide a dynamic set of capabilities to store and exchange data related to blockchain-based identity.



The code for Aries is available at <https://github.com/hyperledger/aries>

Transact

Transact provides a shared library that handles smart contract execution. This library makes the development of distributed ledger software easier by allowing the handling of scheduling, transaction dispatch, and state management via a shared software library. It provides an approach to implement new smart contract development languages named **smart contract engines**. Smart contract engines implement virtual machines or interpreters for smart contract code. Two main examples of such engines are **Seth** (for EVM smart contracts) and **Sabre** (for web assembly-based smart contracts).



Transact is available at <https://crates.io/crates/transact>

Ursa

Ursa is a shared cryptography library that can be used by any project, Hyperledger or otherwise, to provide cryptography functionality. Ursa provides a C-callable library interface and a Rust crate (library). There are two sub-libraries available in Ursa: **LibUrsa** and **LibZmix**. LibUrsa is designed for cryptographic primitives such as digital signatures, standard encryption schemes, and key exchange schemes. LibZmix provides a generic method to produce zero-knowledge proofs. It supports signature **Proofs of Knowledge (PoK)**, bullet proofs, range proofs, and set memberships.



Ursa is available at <https://github.com/hyperledger/ursa>

AnonCreds

AnonCreds stands for **Anonymous Credentials**. This is another very interesting project under incubation at Hyperledger. This is a type of verifiable credential that enables privacy protection based on zero-knowledge proofs.

We've now covered three library projects currently available under Hyperledger. In the next section, we'll explore different tools that can be used in Hyperledger projects.

Tools

There are several projects under the tools category in Hyperledger. Tools under Hyperledger focus on providing utilities and software tools that help to enhance the user experience. For example, visualization tools such as blockchain explorers, deployment tools, and benchmarking tools fall into this category. We'll describe some of these briefly.

Cello

The aim behind Cello is to allow the easy deployment of blockchains. This will provide an ability to allow as a service deployment of a blockchain service. Currently, this project is in the incubation stage.



The source code for Cello is available at <https://github.com/hyperledger/cello>

Caliper

Caliper is a benchmarking framework for blockchains. It can be used to measure the performance of any blockchain. There are different performance indicators supported in the framework. These include **success rate**, **transaction read rate**, **throughput**, **latency**, and **hardware resource consumption**, such as CPU, memory, and I/O. Ethereum, Fabric, Sawtooth, Burrow, and Iroha are currently the supported blockchains in Caliper.



Caliper is available at <https://github.com/hyperledger/caliper>

With this, we complete the introduction to the Hyperledger project's category of tools. Next, we'll introduce domain-specific projects.

Domain-specific

Under Hyperledger, there are also some domain-specific projects that are created to address specific requirements. The project we will explore here is called Grid.

Grid

Grid is a Hyperledger project that provides a standard reference implementation of supply chain-related data types, data models, and relevant business logic-encompassing smart contracts. It is currently in the incubation stage.



The code for Grid is available at <https://github.com/hyperledger/grid>

Each of the projects is in various stages of development. This list is expected to grow as more and more members join the Hyperledger project and contribute to the development of blockchain technology.


Now, in the next section, we will examine the reference architecture of Hyperledger, which provides general principles and design philosophies that can be followed to build new Hyperledger projects.

Hyperledger reference architecture

Hyperledger aims to build new blockchain platforms that are driven by industry use cases. As there have been many contributions made to the Hyperledger project by the community, the Hyperledger blockchain platform is evolving into a protocol for business transactions. Hyperledger is also evolving into a specification that can be used as a reference to build blockchain platforms, as compared to earlier blockchain solutions that address only a specific type of industry or requirement.

In this section, a reference architecture model is presented that has been published by the Hyperledger project. This architecture can be used by a blockchain developer to build a blockchain that is in line with the specifications of the Hyperledger architecture.

Hyperledger has published a whitepaper that presents a reference architecture model that can serve as a guideline to build permissioned distributed ledgers. The reference architecture consists of various components that form a business blockchain.



Various whitepapers related to the creation and design of Hyperledger are available here:
<https://www.hyperledger.org/learn/white-papers>

These high-level components are shown in the reference architecture diagram here, which has been drawn from the whitepaper:

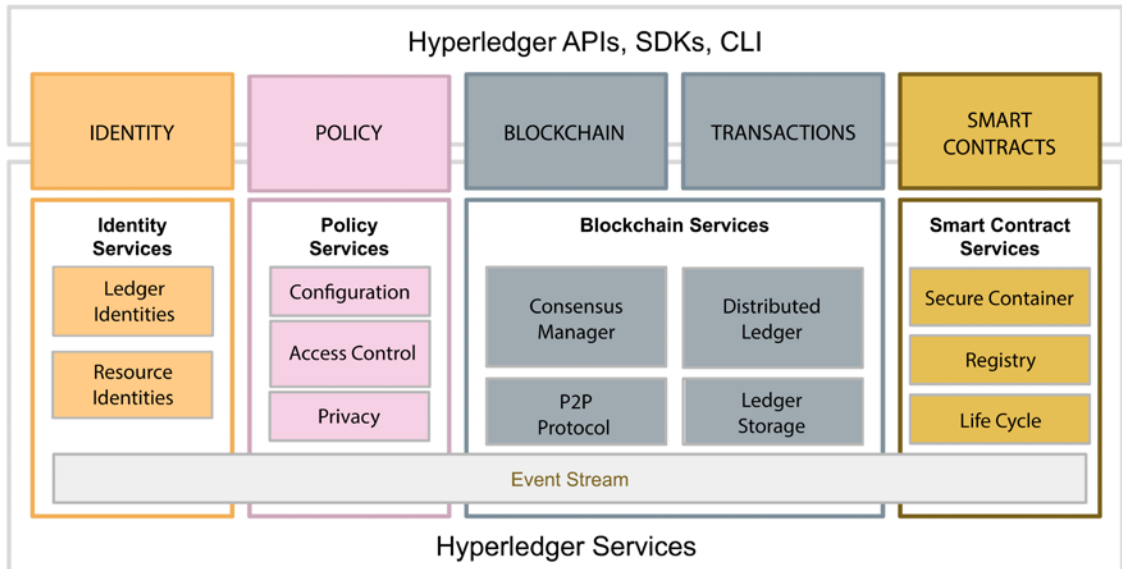


Figure 14.1: Reference architecture

In the preceding diagram, starting from the left, we see that we have five top-level components that provide various services. The first is **identity**, which provides authorization, identification, and authentication services under membership services. Then, we have the **policy** component, which provides policy services.

After this, we see **blockchain** and **transactions**, which consist of the **Consensus Manager**, **Distributed Ledger**, the network **P2P Protocol**, and **Ledger Storage** services. The consensus manager ensures that the ledger is updateable only through consensus among the participants of the blockchain network.

Finally, we have the **smart contracts** layer, which provides chaincode services in Hyperledger and makes use of **Secure Container** technology to host smart contracts. Chaincode can be considered the Hyperledger equivalent of smart contracts.



Chaincode is a program that implements a specific interface. It is usually written in Java, Node.js, or Go. Even though the terms *smart contract* and *chaincode* are used interchangeably in Hyperledger Fabric, there is a subtle difference between chaincode and smart contract. A smart contract can be defined as a piece of code that defines the transaction logic, which controls the transaction lifecycle and can result in updating the world state. Chaincode is a relevant but slightly different concept—it is a deployable object that contains smart contracts packaged within it. A single chaincode can contain multiple smart contracts and after deployment, all smart contracts contained within the chaincode are available for use. Generally, however, the terms are used interchangeably.

We will see all these in more detail in the *Hyperledger Fabric* section shortly.

From a components perspective, Hyperledger contains various elements, as described here:

- **Consensus:** These services are responsible for facilitating the agreement process between the participants on the blockchain network. Consensus is required to make sure that the order and state of transactions are validated and agreed upon in the blockchain network.
- **Smart contracts:** These services are responsible for implementing business logic as per the requirements of the users. Transactions are processed based on the logic defined in the smart contracts that reside on the blockchain.
- **Communication:** This component is responsible for message transmission and exchange between the nodes on the blockchain network.
- **Security and crypto:** These services are responsible for providing the capability to allow various cryptographic algorithms or modules to provide privacy, confidentiality, and non-repudiation services.
- **Data store:** This component provides an ability to use different data stores for storing the state of the ledger. This means that data stores are also pluggable, allowing the usage of any database backend, such as couchdb or goleveldb.
- **Policy services:** This set of services provides the ability to manage the different policies required for the blockchain network. This includes endorsement policy and consensus policy.
- **APIs and SDKs:** This component allows clients and applications to interact with the blockchain. An SDK is used to provide mechanisms to deploy and execute chaincode, query blocks, and monitor events on the blockchain.

In the next section, we are going to discuss the design goals of Hyperledger.

Hyperledger design principles

There are certain requirements for a blockchain service. The reference architecture is driven by the needs and requirements raised by the participants of the Hyperledger project after studying the industry use cases. There are several categories of requirements that have been deduced from the study of industrial use cases and are seen as principles of design philosophy. We'll describe these principles in the following sections.

- **Modular structure:** The main requirement of Hyperledger is a modular structure. It is expected that a cross-industry blockchain will be used in many business scenarios, and as such, functions related to storage, policy, chaincode, access control, consensus, and many other blockchain services should be modular and pluggable. The specification provided in the Hyperledger reference architecture suggests that the modules should be “plug and play” and users should be able to easily remove and add a different module that meets the requirements of the business.
- **Privacy and confidentiality:** This is one of the most critical factors. As traditional blockchains are permissionless, in a permissioned model it is of the utmost importance that transactions on the network are visible to only those who are allowed to view it. The privacy and confidentiality of transactions and contracts are of absolute importance in a business blockchain. As such, Hyperledger's vision is to provide support for a full range of cryptographic protocols and algorithms. We discussed cryptography in *Chapter 3, Symmetric Cryptography*, and *Chapter 4, Asymmetric Cryptography*.

It is expected that users will be able to choose appropriate modules according to their business requirements. For example, if a business blockchain needs to be run only between already-trusted parties and performs very basic business operations, then perhaps there is no need to have advanced cryptographic support for confidentiality and privacy. Therefore, users should be able to remove that functionality (module) or replace it with a more appropriate module that suits their needs.

Similarly, if users need to run a cross-industry blockchain, then confidentiality and privacy can be of paramount importance. In this case, users should be able to plug an advanced cryptographic and access control mechanism (module) into the blockchain, which can even allow the usage of **hardware security modules (HSMs)**. The blockchain should also be able to handle sophisticated cryptographic algorithms without compromising performance. In addition to the previously mentioned scenarios, due to regulatory requirements in business, there should also be a provision to allow the implementation of privacy and confidentiality policies in conformance with regulatory and compliance requirements.

- **Identity:** To provide privacy and confidentiality services, a flexible **Public Key Infrastructure (PKI)** model that can be used to handle the access control functionality is also required. The strength and type of cryptographic mechanisms are also expected to vary according to the needs and requirements of the users. In certain scenarios, it might be required for a user to hide their identity, and as such, Hyperledger is expected to provide this functionality.
- **Scalability:** This is another major requirement that, once met, will allow reasonable transaction throughput, which will be sufficient for all business requirements and also a large number of users.

- **Deterministic transactions:** This is a core requirement in any blockchain. If transactions do not produce the same result every time they are executed, then regardless of who and where the transaction is executed, achieving consensus is impossible. Therefore, deterministic transactions become a key requirement in any blockchain network. We discussed these concepts in *Chapter 8, Smart Contracts*.
- **Auditability:** Auditability is another requirement of businesses. It is expected that an immutable audit trail of all identities, related operations, and any changes is kept.
- **Interoperability:** Currently, there are many blockchain platforms available, but they cannot communicate with each other easily. This can be a limiting factor in the growth of a blockchain-based global business ecosystem. It is envisaged that many blockchain networks will operate in the business world for specific needs, but it is important that they are able to communicate with each other. There should be a common set of standards that all blockchains can follow to allow communication between different ledgers. There are different efforts already underway to achieve this, not only under the Hyperledger umbrella, such as Hyperledger Quilt, but also other projects such as Cosmos and Polkadot.
- **Portability:** The portability requirement is concerned with the ability to run across multiple platforms and environments without the need to change anything at the code level. Hyperledger Fabric, for example, is envisaged to be portable, not only at the infrastructure level, but also at the code, library, and API levels, so that it can support uniform development across various implementations of Hyperledger.
- **Rich data queries:** The blockchain network should allow rich queries to be run on the network. This can be used to query the current state of the ledger using traditional query languages, such as SQL, and supporting statements like SELECT, which will allow wider adoption and ease of use.

All the points describe the general guiding principles that allow the development of blockchain solutions that are in line with the Hyperledger design philosophy.

In the next section, we will look at Hyperledger Fabric in detail, which is the first project to graduate to active status under Hyperledger.

Hyperledger Fabric

Hyperledger Fabric, or **Fabric** for short, is the contribution made initially by IBM and digital assets to the Hyperledger project. This contribution aims to enable a modular, open, and flexible approach toward building blockchain networks.

Key concepts

Various functions in the Fabric are pluggable, and it also allows the use of any language to develop smart contracts. This functionality is possible because it is based on container technology (Docker), which can host any language.

Chaincode is sandboxed in a secure container, which includes a secure operating system, the chaincode language, runtime environment, and SDKs for Go, Java, and Node.js. Other languages could be supported too in the future, if required, but this needs some development work.

This ability is a compelling feature compared to domain-specific languages in Ethereum, or the limited scripted language in Bitcoin. It is a permissioned network that aims to address issues such as scalability, privacy, and confidentiality. The fundamental idea behind this is modularization, which would allow flexibility in the design and implementation of the business blockchain. This can then result in achieving scalability, privacy, and other desired attributes and fine-tuning them according to requirements.

Transactions in Fabric are private, confidential, and anonymous for general users, but they can still be traced back and linked to the users by authorized auditors. As a permissioned network, all participants are required to be registered with the membership services to access the blockchain network. This ledger also provides an auditability functionality to meet the regulatory and compliance needs required by the user. There are six core capabilities of the Hyperledger Fabric blockchain:

- Modular architecture that makes Hyperledger Fabric resilient to changes and allows any industry to adopt as required.
- Privacy and confidentiality that allows sharing of data only between specific subsets of network members by using private channels.
- Efficiency and scalability in the Hyperledger Fabric network is provided due to Fabric's ability to assign roles to nodes, which results in concurrency and parallelism. Transactions can be threaded for faster processing in Hyperledger Fabric. Also, as execution can only be required at a specific subset of nodes, some network resources can remain free for performing other tasks, thus resulting in a more efficient network.
- Business logic in Fabric is provided by smart contracts called chaincode.
- Identity management is provided using a **membership service provider (MSP)**, which manages user IDs and provides authentication services for users on the network.
- Governance is an important capability that enables policy building and enforcement to ensure access control security of a system. For example, a policy is required to identify the participants that are allowed to deploy a chaincode or make configuration changes.



Note that the Hyperledger reference architecture described earlier is not necessarily strictly followed in every Hyperledger project. It is there as a reference and guideline to follow; however, not all elements of the reference architecture are implemented in a Hyperledger project.

Hyperledger Fabric is composed of several modules that provide various services. Now we discuss these services in detail.

Membership service

This service is used to provide access control capability for the users of the Fabric network. It provides identity management capabilities and allows for custom and complex access control policies. Membership services perform the following functions:

- User identity verification
- User registration

- Assign appropriate permissions to the users depending on their roles

Membership services make use of a **certificate authority (CA)** to support identity management and authorization operations. This CA can be internal (such as Fabric CA, which is a default interface in Hyperledger Fabric), or an organization can opt to use an external certificate authority. Fabric CA issues **enrolment certificates (E-Certs)**, which are produced by an **enrolment certificate authority (E-CA)**. Once peers are issued with an identity, they are allowed to join the blockchain network. There are also temporary certificates issued called **T-Certs**, which are used for one-time transactions.

All peers and applications are identified using a certificate authority. An authentication service is provided by the certificate authority. MSPs can also interface with existing identity services like the **Lightweight Directory Access Protocol (LDAP)**. The default implementation of MSP in Hyperledger Fabric is based on a widely used internet standard specifying the use and structure of public key certificates, the X.509 standard. This means that Hyperledger Fabric can support any PKI where certificates are issued by standard certificate authorities on the internet.

Hyperledger Fabric is also fortified with a cryptographic protocol called **identity mixer**, which provides strong authentication while providing identity privacy.

A closely associated concept is **policies**. Policies control who is allowed to do what and on which component of the Hyperledger network. While MSPs recognize valid identities, policies implement permissions for these identities.

This section covered the membership services implemented in Hyperledger Fabric. Next, we'll introduce some of Fabric's blockchain services.

Blockchain services

Blockchain services are at the core of Hyperledger Fabric. Components within this category are as follows.

Consensus services

A consensus service is responsible for providing the interface to the consensus mechanism. This serves as a module that is pluggable and receives the transaction from other Hyperledger entities and executes them under criteria, according to the type of mechanism chosen.

Consensus in Hyperledger Fabric is implemented as a peer called **orderer**, which is responsible for ordering the transactions in sequence into a block. An orderer does not hold smart contracts or ledgers. Consensus is pluggable and currently ordering services in Hyperledger Fabric are based on a consensus mechanism called RAFT, which is a crash fault-tolerant and leader-follower-based protocol for achieving distributed consensus.



We discussed consensus mechanisms in appropriate detail in *Chapter 5, Consensus Algorithms*. Readers can refer to this chapter to review the RAFT and PBFT mechanisms.

In addition to these mechanisms, other mechanisms may become available in the future that can be plugged into Hyperledger Fabric.

Distributed ledger

Blockchain and world state are two main elements of the distributed ledger. Blockchain is simply a cryptographically linked list of blocks (as introduced in *Chapter 1, Blockchain 101*) and world state is a key-value database that stores the current values of a set of ledger states. World state makes it easier to get this latest state instead of programs being required to traverse the entire blockchain database. This database is used by smart contracts to store relevant states during execution of the transactions. A blockchain consists of blocks that contain transactions. These transactions contain chaincode, which runs transactions that can result in updating the world state. Each node saves the world state on disk in LevelDB or CouchDB, depending on the implementation. As Fabric allows pluggable data stores, you can choose any data store for storage.

A block consists of three main components called **block header**, **block data (transactions)**, and **block metadata**. The following diagram shows a blockchain depiction with the block and transaction structure in Hyperledger Fabric, with the relevant fields:

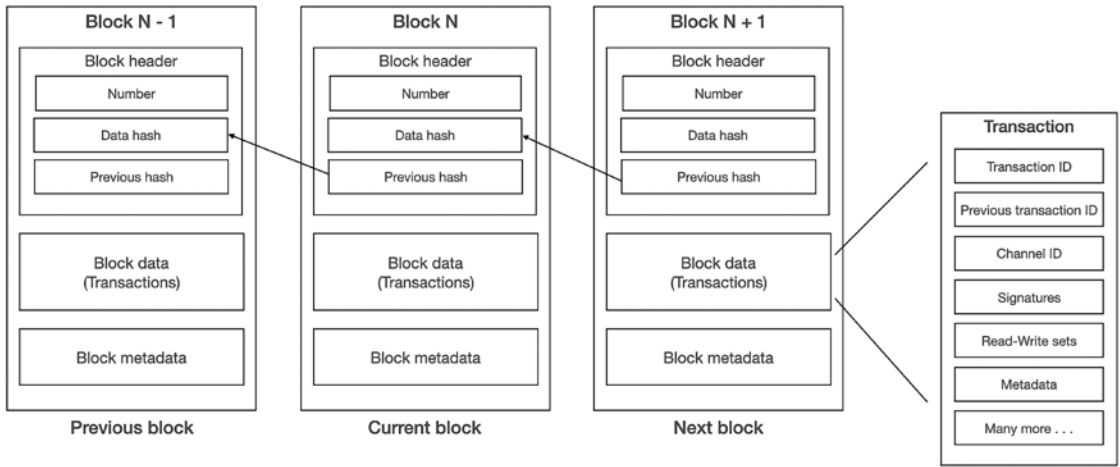


Figure 14.2: Blockchain and transaction structure

Block header consists of three fields, namely **Number**, **Data hash**, and **Previous hash**.

Block data contains an ordered list of transactions. A **Transaction** is made up of multiple fields such as **Transaction ID**, **Previous transaction ID**, **Channel ID**, **Signatures**, **Read-Write sets**, **Metadata**, and many more.

Block metadata mainly consists of the creator identity, the time of creation, and relevant signatures.

The peer-to-peer protocol

The P2P protocol in Hyperledger Fabric is built using **Google RPC (gRPC)**. It uses protocol buffers to define the structure of the messages.

Messages are passed between nodes to perform various functions. There are four main types of messages in Hyperledger Fabric: **discovery**, **transaction**, **synchronization**, and **consensus**. Discovery messages are exchanged between nodes when starting up to discover other peers on the network. Transaction messages are used to deploy, invoke, and query transactions, and consensus messages are exchanged during consensus. Synchronization messages are passed between nodes to synchronize and keep the blockchain updated on all nodes.

Ledger storage

To save the state of the ledger, by default, LevelDB is used, which is available at each peer. An alternative is to use CouchDB, which provides the ability to run rich queries.

Now that we've established some of the core blockchain services used in Hyperledger Fabric, let's look at some of the chaincode, or smart contract, services available, along with an overview of the APIs and CLIs available in Fabric.

Smart contract services

These services allow the creation of secure containers that are used to execute the chaincode. Components in this category are as follows:

- **Secure container:** Chaincode is deployed in Docker containers that provide a locked-down sandboxed environment for smart contract execution. Currently, Golang is supported as the main smart contract language, but any other mainstream languages can be added and enabled if required.
- **Secure registry:** This provides a record of all images containing smart contracts.
- **Events:** Events on the blockchain can be triggered by endorsers and smart contracts. External applications can listen to these events and react to them if required via event adapters. They are similar to the concept of events introduced in Solidity in *Chapter 11, Tools, Languages, and Frameworks for Ethereum Developers*.

APIs and CLIs

An application programming interface provides an interface into Fabric by exposing various REST APIs. Additionally, command-line interfaces that provide a subset of REST APIs and allow quick testing and limited interaction with the blockchain are also available.

In the preceding sections, we have covered the main services of Hyperledger Fabric. Next, let's consider Hyperledger Fabric's components from a network perspective.

Components

There are various components that can be part of the Hyperledger Fabric blockchain network. These components include, but are not limited to, the peers, clients, channels, world state database, transactions, membership service providers, smart contracts, and crypto-service provider components.

Peers/nodes

Peers participate in maintaining the state of the distributed ledger. They also hold a local copy of the distributed ledger. Peers communicate via gossip protocol. There are three types of peers in the Hyperledger Fabric network:

- **Endorsing peers or endorsers**, which simulate the transaction execution and generate a read-write set. **Read** is a simulation of a transaction's reading of data from the ledger and **write** is the set of updates that would be made to the ledger if and when the transaction is executed and committed to the ledger. Endorsers execute and endorse transactions. It should be noted that an endorser is also a committer too. Endorsement policies are implemented with chaincode and specify the rules for transaction endorsement.
- **Committing peers or committers**, which receive transactions endorsed by endorsers, verify them, and then update the ledger with the read-write set. A committer verifies the read-write set generated by the endorsers along with transaction validation.
- **Orderer nodes** receive transactions from endorsers along with read-write sets, arrange them in a sequence, and send them to committing peers. Committing peers then perform validation and committing to the ledger.

All peers make use of certificates issued by membership services. A peer can assume two roles. A peer can be a leader peer or an anchor peer. In a channel when an organization has multiple peers, a leader peer is responsible for transaction distribution from the orderer node to the other committer nodes in the organization. An anchor peer is used when a peer needs to communicate with a peer in another organization. A peer can assume committer, endorser, leader, and anchor roles at the same time.

Clients

Clients are software that makes use of APIs to interact with Hyperledger Fabric and propose transactions.

Channels

Channels allow the flow of confidential transactions between different parties on the network. They allow the use of the same blockchain network but with separate overlay blockchains. Channels allow only members of the channel to view the transactions related to them; all other members of the network will not be able to view the transactions. We can think of a channel as a private subnet for communication between two or more network members.

World state database

World state reflects all the committed transactions on the blockchain. This is essentially a key-value store that is updated because of transactions and chaincode execution. For this purpose, either LevelDB or CouchDB is used. LevelDB is a key-value store whereas CouchDB stores data as JSON objects, which allows rich queries to run against the database. A related concept is private data, which enables privacy in the Hyperledger blockchain.

We can see this concept visually in the diagram below:

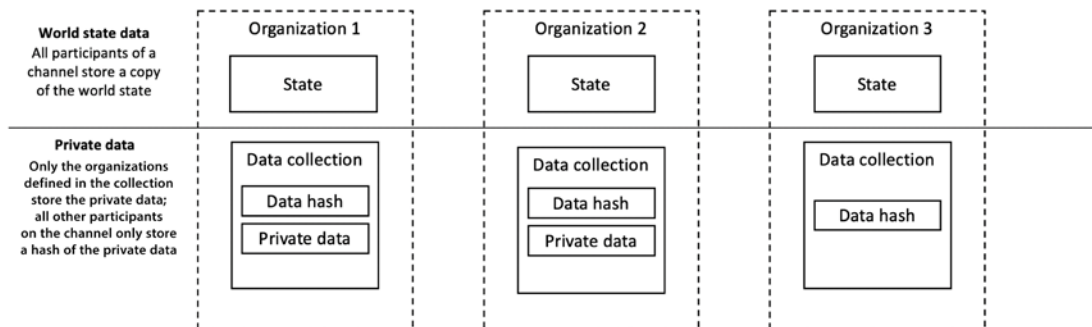


Figure 14.3: World state and private data

The preceding diagram shows a channel shared between multiple organizations. Note that while the “state” is shared between all organizations, the private data is only available to those peers and organizations that are party to the transaction. Peers that do not need to know the transaction only have a hash of the private data. This concept leads us to **private data collections (PDCs)**.

Private data collections

PDCs are associated with organizations. They define how private data should be disseminated and endorsed, and are part of the chaincode definition. They are composed of a private section and a public section. The private section consists of data that is disseminated through a P2P protocol between the organizations’ peers that belong to the PDC. The public section contains hashes of the private data, which are endorsed and committed to the ledger. PDCs are useful in scenarios where an ordering service cannot be trusted for confidentiality and when data in a single ledger or a Fabric channel is made visible to only specific parties, i.e., restricted private transactions.

Transactions

Transaction messages can be divided into two types: **deployment transactions** and **invocation transactions**. The former is used to deploy a new chaincode to the ledger, and the latter is used to call functions from the smart contract. Transactions can be either public or confidential. Public transactions are open and available to all participants, while confidential transactions are visible only in a channel open to its participants.

Membership Service Provider

The **MSP** is a modular component that is used to manage identities on the blockchain network. This provider is used to authenticate clients who want to join the blockchain network. A **certificate authority (CA)** is used in the MSP to provide identity verification and binding services.

Smart contracts

We discussed smart contracts in detail in *Chapter 8, Smart Contracts*. In Hyperledger Fabric, the same concept of smart contracts is implemented, but they are called chaincode instead of smart contracts. They contain conditions and parameters to execute transactions and update the ledger. Chaincode is usually written in Golang or Java.

Crypto service provider

As the name suggests, this is a service that provides cryptographic algorithms and standards for use in the blockchain network. This service provides key management, signature and verification operations, and encryption-decryption mechanisms. This service is used with the membership service to provide support for cryptographic operations for elements of blockchain, such as endorsers, clients, and other nodes and peers.

After this introduction to some of the components of Hyperledger Fabric, we will next see what an application looks like when on a Hyperledger network.

Applications

A typical application on Fabric is simply composed of a user interface, usually written in JavaScript/HTML, that interacts with the backend chaincode (smart contract) stored on the ledger via an API layer:

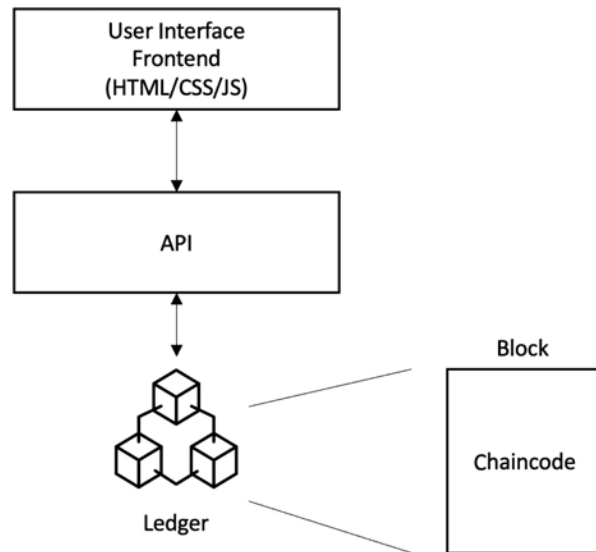


Figure 14.4: A typical Fabric application

Hyperledger provides various APIs and command-line interfaces to enable interaction with the ledger. These APIs include interfaces for identity, transactions, chaincode, ledger, network, storage, and events.

In the next section, we'll see how chaincode is implemented.

Chaincode implementation

Chaincode is usually written in Golang or Java. Chaincode can be public (visible to all on the network), confidential, or access controlled. These code files serve as a smart contract that users can interact with via APIs. Users can call functions in the chaincode that result in a state change, and consequently update the ledger.

There are also functions that are only used to query the ledger and do not result in any state change. Chaincode implementation is performed by first creating the chaincode shim interface in the code. Shim packages provide APIs for accessing state variables, the transaction context of chaincode, and call other chaincodes. They can either be in Java or Golang code.

The following four functions are required in order to implement the chaincode:

- `Init()`: This function is invoked when the chaincode is deployed onto the ledger. This initializes the chaincode and results in making a state change, which updates the ledger accordingly.
- `Invoke()`: This function is used when contracts are executed. It takes a function name as parameters along with an array of arguments. This function results in a state change and writes to the ledger.
- `Query()`: This function is used to query the current state of a deployed chaincode. This function does not make any changes to the ledger.
- `Main()`: This function is executed when a peer deploys its own copy of the chaincode. The chaincode is registered with the peer using this function.

The following diagram illustrates the general overview of Hyperledger Fabric. Note that peer clusters at the top include all types of nodes: **Endorsers**, **Committers**, and **Orderers**:

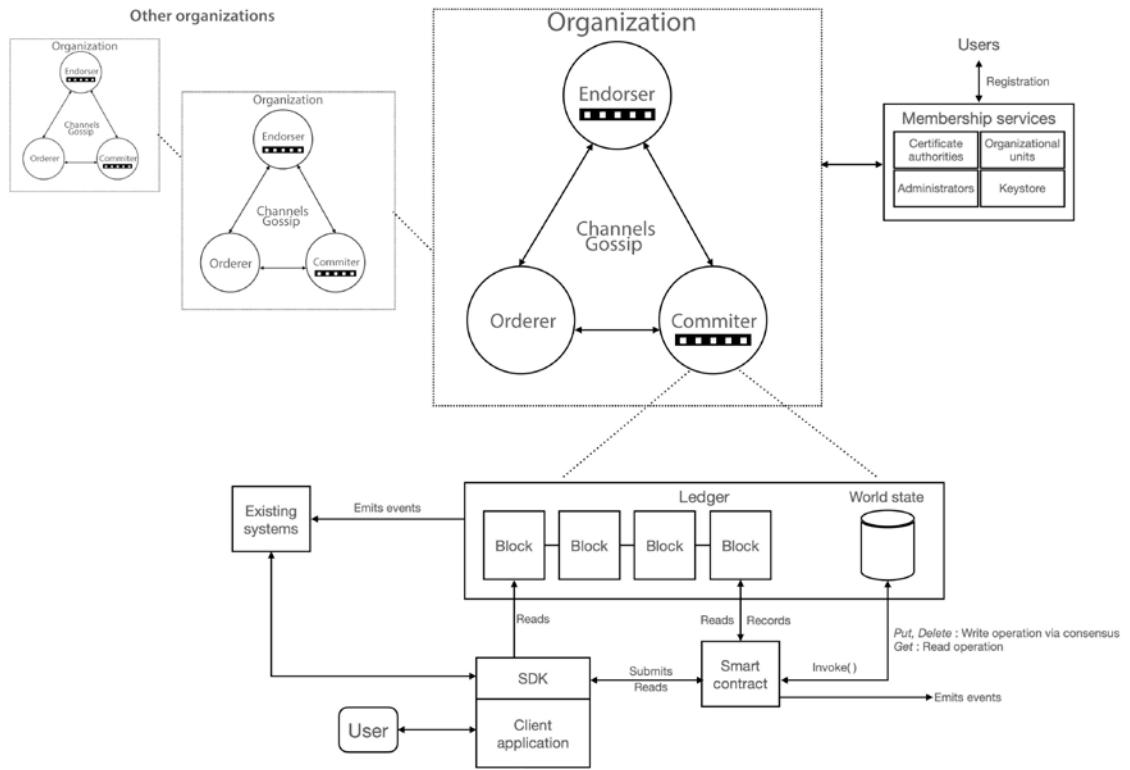


Figure 14.5: A high-level overview of a Hyperledger Fabric network

The preceding diagram shows that peers (shown center-top) communicate with each other, and each node has a copy of the blockchain. In the top-right corner, the membership services are shown, which validate and authenticate peers on the network by using a CA. At the bottom of the diagram, a magnified view of the blockchain is shown whereby existing systems can produce events for the blockchain and can also listen for the blockchain events, which can then optionally trigger an action. At the bottom right-hand side, a user's interaction with the application is shown. The application in turn interacts with the smart contract via the `Invoke()` method, and smart contracts can query or update the state of the blockchain.

Chaincode in Hyperledger Fabric doesn't need to be deterministic and can be developed in any mainstream language, such as JavaScript, Java, or Golang.

In this section, we saw how chaincode is implemented and examined the high-level architecture of Hyperledger Fabric. In the next section, we will discuss the application model.

The application model

Any blockchain application for Hyperledger Fabric follows the MVC-B architecture. This is based on the popular MVC design pattern. Components in this model are **View**, **Control**, **Model** (the data model), and **Blockchain**:

- **View logic:** This is concerned with the user interface. It can be a desktop, web application, or a mobile frontend.
- **Control logic:** This is the orchestrator between the user interface, data model, and APIs.
- **Data model:** This model is used to manage the off-chain data.
- **Blockchain logic:** This is used to manage the blockchain via the controller and the data model via transactions.



The IBM Cloud service offers sample applications for blockchain under its blockchain as a service offering. It is available at <https://www.ibm.com/blockchain/platform/>. This service allows users to create their own blockchain networks in an easy-to-use environment.

After this brief introduction to the application model, let's move on to another important topic, consensus, which is important not only in Hyperledger Fabric but is also central to blockchain design and architecture.

Consensus mechanism

The consensus mechanism in Hyperledger Fabric consists of three steps:

- **Transaction endorsement:** This process endorses the transactions by simulating the transaction execution process.
- **Ordering:** This is a service provided by the cluster of **orderers that run the consensus algorithm**, which takes endorsed transactions and decides on a sequence in which the transactions will be written to the ledger.
- **Validation and commitment:** This process is executed by committing peers, which first validate the transactions received from the orderers and then commit that transaction to the ledger.

These steps are shown in the following diagram:

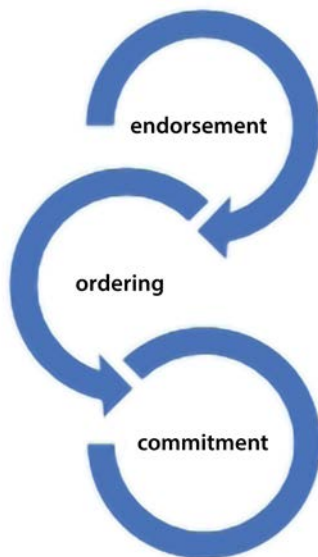


Figure 14.6: The consensus flow

The ordering step is also pluggable. There are several ordering services available for the Fabric consensus mechanism. Initially, in some versions of Fabric 1.0, only simple ordering services such as Apache Kafka and SOLO were available. However, now, other advanced consensus algorithms such as RAFT are also available.

In the next section, we'll describe how a transaction flows through various stages in Hyperledger Fabric before finally updating the ledger.

Transaction lifecycle

There are several steps that are involved in a transaction flow in Hyperledger Fabric. The steps are described in detail as follows:

1. Transaction proposal by clients. This is the first step where a transaction is proposed by the clients and sent to endorsing peers on the distributed ledger network. All clients need to be enrolled via membership services before they can propose transactions.
2. The transaction is simulated by endorsers and generates a **read-write (RW)** set. This is achieved by executing the chaincode, but instead of updating the ledger, only an RW set depicting any reads or updates to the ledger is created.

3. The endorsed transaction is sent back to the application.
4. The endorsed transactions and RW sets are submitted to the ordering service by the application.
5. The ordering service assembles all endorsed transactions and RW sets in order in a block and sorts them by channel ID.
6. The ordering service broadcasts the assembled block to all committing peers. Committing peers take the following actions:
 1. Validate the endorsement policy.
 2. Validate read set versions in the state DB.
 3. Commit the block to the blockchain.
 4. Commit the valid transaction to the state DB.
7. The validation ensures two conditions are met: first, transactions are executed as per the given transaction logic and secondly, there are no state conflicts in the submitted transactions, e.g., a scenario where two transactions are updating the same state.
8. Finally, notification of success or failure of the transaction by committing peers is sent back to the clients/applications.

The following diagram represents the steps and the Fabric architecture from a transaction flow perspective:

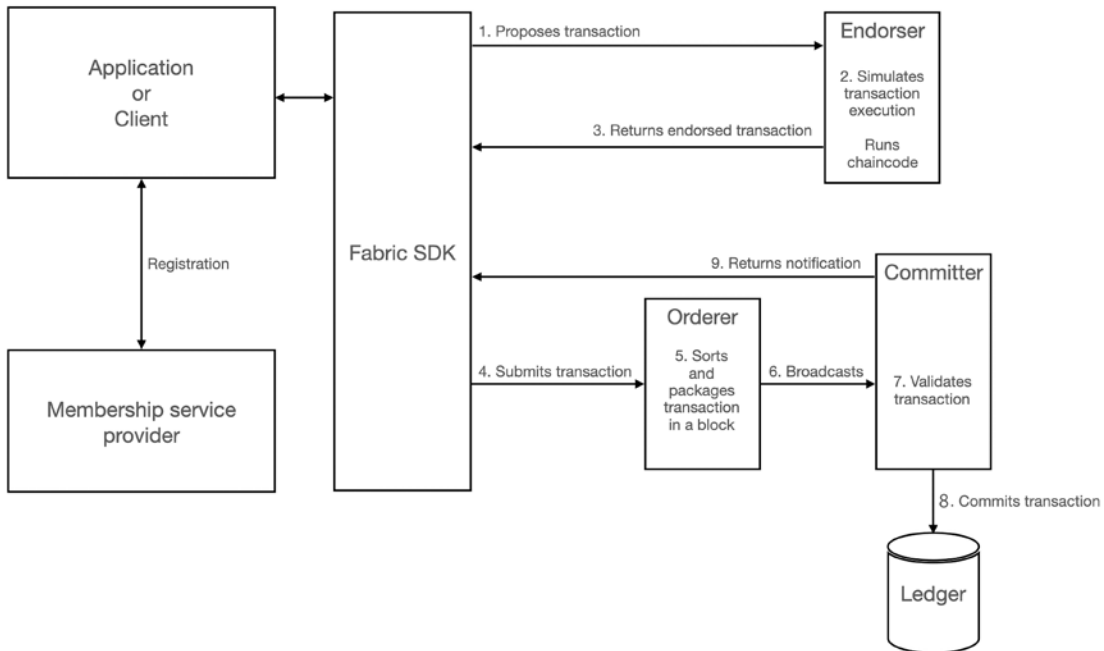


Figure 14.7: The transaction flow

As you can see in the preceding diagram, the first step is to propose transactions, which a client does via an SDK. Before this, it is assumed that all clients and peers are registered with the membership service provider.

Hyperledger Fabric uses an execute-order-validate pattern that allows transactions to execute before the blockchain achieves consensus, which enables better efficiency. In Fabric every transaction is executed and endorsed only by a subset of peers, which allows parallel execution. Also, the final state of a transaction is only written to the blockchain.

So far, we have covered the core architecture of Hyperledger Fabric, which is the most popular Hyperledger project.

In the next section, we will introduce the new features introduced with Fabric 2.0.

Fabric 2.0

This is a major upgrade to the protocol. The fundamentals remain the same, but some very interesting new features and improvements have been made, which are introduced in the following sections.



Hyperledger Fabric 2.0 was released on January 30, 2020. The latest release is available at <https://github.com/hyperledger/fabric/releases/latest>

New chaincode lifecycle management

In Hyperledger Fabric 2.0, new decentralized chaincode lifecycle management is implemented, which is the main notable difference between Fabric 1.x and Fabric 2.x. In the former, an organization administrator can install the chaincode in its own organization, and other organizations' administrators have to unconditionally agree to instantiate (deploy or upgrade) the chaincode. In other words, there is no agreement process between organizations to agree on the installation of a chaincode. This problem is resolved by introducing checks, which enable peers to only participate in a chaincode if their organizational administrators have agreed to it.

First, let's look at the chaincode lifecycle in Fabric 1.0:

1. Each organization administrator installs chaincode on their peers.
2. An administrator instantiates the chaincode by deploying or upgrading.
3. After results are received, the transaction is submitted to the orderers.
4. After ordering, it is submitted to all peers and committed.

Now, we'll look at Fabric 2.0's lifecycle, which highlights the differences between it and Fabric 1.0:

1. Install chaincode by putting the package on the filesystem. Each organization administrator installs chaincode on their peers.
2. Provide approval by executing the `approveformyorg` function. This verifies that the organization agrees with the chaincode definition.

3. Send the ordering service to the orderer.
4. Commit to the organization collection. The collection is a private data collection that provides a mechanism to allow private data sharing only between those organizations that are party to the transaction, even if they are within the same channel. In other words, collections allow private data sharing between a subset of organizations on a channel. Even though other organizations are members of the same channel, they cannot see the data shared between a subset of organizations that are sharing data using the collection, also called a private data collection.
5. Other organizations' administrators perform the same *steps 1 to 4*.
6. After these steps are executed, there is a record in each organization indicating the agreement on the chaincode definition.
7. Define the chaincode as per the agreement in *step 6*.
8. Submit to ordering.
9. Commit to the definition of all peers.
10. The organization administrator who committed the chaincode definition now invokes the `init` function on its own peers and other organizations' peers.
11. Submit to ordering.
12. Commit all the peers.



Note that Fabric 2.0 supports both legacy and new lifecycle patterns, but eventually the legacy lifecycle will become deprecated.

In addition to the chaincode lifecycle management changes in Fabric 2.0, there are also new chaincode application patterns. We will introduce these next.

New chaincode application patterns

There are also new chaincode application patterns introduced in Fabric 2.0 for collaboration and consensus. These are automated checks that can be added by an organization to enable additional validation before a transaction is endorsed. In addition, a decentralized agreement is also supported where multiple organizations can propose conditions for an agreement, and when those conditions are met, a collective agreement can be made on a transaction.

- **Enhanced data privacy:** Fabric 2.0 enhances data privacy by introducing collection-level endorsement policies, per-organization collections, and the flexible sharing and verification of private data.
- **External chaincode launcher:** In Hyperledger Fabric 2.0, by default, Docker APIs are used to build and run chaincode. In addition, external builders and launchers can be used if required. In previous versions, there was a requirement to have access to the Docker daemon to build and run chaincode, but in Fabric 2.0 there is no such dependency. There is also no requirement to run chaincode in Docker containers anymore; chaincode can be run in any environment deemed appropriate by the network operator.

- **RAFT consensus:** RAFT is a **crash fault-tolerant (CFT)** ordering service for achieving consensus. It is based on the etcd library and a leader-follower model. In this model, a leader is elected whose decision is then replicated by its followers.
- **Better performance:** In Fabric 2.0, caching has been introduced at the CouchDB state database level to improve performance by reducing the read latency, introduced due to expensive lookup operations in previous versions.

With all that we have learned so far, the Hyperledger application architecture can be depicted as follows:

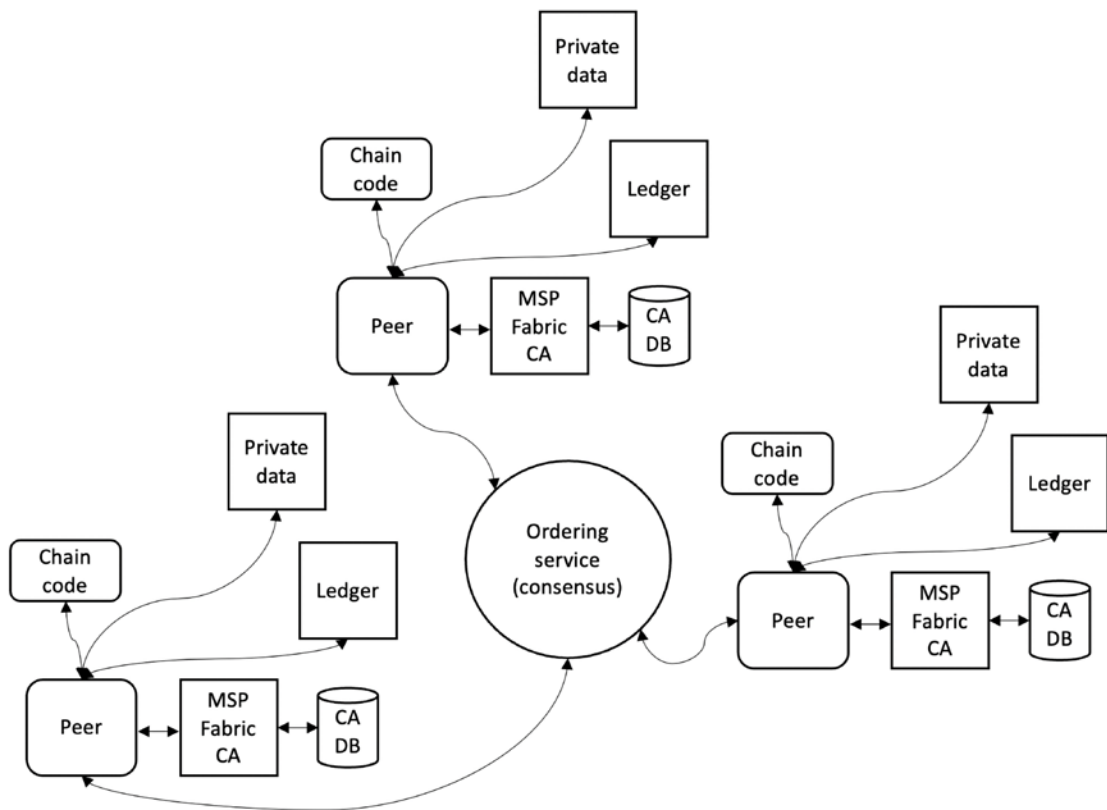


Figure 14.8: Hyperledger Fabric application architecture



A list of cross-industry projects built with Hyperledger Fabric and other Hyperledger projects is being maintained here: <https://www.hyperledger.org/learn/blockchain-showcase>. Readers can explore this further to understand how Hyperledger Fabric is used in different use cases.

With this section, our introduction to Hyperledger Fabric is complete. Hyperledger Fabric is the flagship project of Hyperledger and is used in a wide variety of applications. Hyperledger Fabric is continually evolving, and more features and changes are expected in future releases. However, no major design changes are expected since the introduction of version 2.0.

Summary

In this chapter, we have gone through an introduction to the Hyperledger project. Firstly, the core ideas behind the Hyperledger project were discussed, and a brief introduction to all projects under Hyperledger was provided. A key Hyperledger project was discussed in detail, namely, Hyperledger Fabric. All these projects are continuously improving, and changes are expected in the next releases. However, the core concepts of all the projects mentioned previously are expected to remain unchanged or change only slightly. Readers are encouraged to visit the relevant links provided in the chapter to see the latest updates.

It is evident that a lot is going on in this space, and projects like Hyperledger from the Linux Foundation are playing a pivotal role in the advancement of blockchain technology. Each of the projects introduced in this chapter has novel approaches toward solving the issues faced in various industries, and any current limitations within blockchain technology are also being addressed, such as scalability and privacy. It is expected that more projects will soon be proposed for the Hyperledger project, and it is envisaged that with this collaborative and open effort, blockchain technology will advance tremendously and collectively benefit the community.

In the next chapter, we will discuss tokenization, which is one of the most prominent applications of blockchain technology and is disrupting many industries, especially the financial industry.

Join us on Discord!

To join the Discord community for this book – where you can share feedback, ask questions to the author, and learn about new releases – follow the QR code below:



<https://packt.link/ips2H>